Available online at www.sciencedirect.com

SCIENCE DIRECT®

LINEAR ALGEBRA AND ITS APPLICATIONS

ELSEVIER Linear Algebra and its Applications 366 (2003) 121–138

www.elsevier.com/locate/laa

An application of fast factorization algorithms in Computer Aided Geometric Design☆

G. Casciola [a,*], F. Fabbri [b], L.B. Montefusco [a]

[a] *Department of Mathematics, University of Bologna, P.zza di Porta S.Donato, 5, 40127 Bologna, Italy*
[b] *Department of Mathematics, University of Padova, Via Belzoni, 7, 35131 Padova, Italy*

Received 25 May 2001; accepted 6 June 2002

Submitted by D.A. Bini

Abstract

Structured matrices play an important role in the numerical solution of practical problems, because it is possible to develop fast algorithms for their triangular factorization. In this paper we consider a classical problem of Computer Aided Geometric Design, namely the computation of the intersection points of two planar rational parametric curves, given in Bernstein form. For the numerical solution to this problem we propose an algebraic approach, based on a fast factorization algorithm of the resulting Bezout matrix with polynomial entries, which avoids the need for symbolic computation. This also allows us to efficiently handle high degree curves. Numerical examples and comparisons with other standard intersection methods are given.

AMS classification: 15A23; 15A36; 65D18; 65Y20

Keywords: Structured matrices; Bezout resultant; Curve intersection

1. Introduction

The problem of computing the intersection of parametric curves arises in many applications of computer graphics and solid modeling. For example, intersection

☆ This research was supported by MURST, Cofin2000 and 60% projects, by University of Bologna "Funds for selected research topics, and Grant CNR no. 99.1707".

* Corresponding author.

E-mail addresses: casciola@dm.unibo.it (G. Casciola), ffabbri@csr.unibo.it (F. Fabbri), montelau@dm.unibo.it (L.B. Montefusco).

0024-3795/03/$ - see front matter
doi:10.1016/S0024-3795(02)00469-X

is the basis of hidden-curve removal algorithms for free-form surfaces and is a basic step in performing Boolean operations on boundary representation in a CAD system.

Methods for computing the intersection of rational parametric curves have been extensively studied in the literature and are essentially based on algebraic or geometric approaches [7,8]. A comparative performance evaluation [12] has shown that the methods based on implicitization are faster than other intersection algorithms for curves of degree up to 4, but their performance degrades for higher degree curves. This can be explained by recalling that such methods are based on the symbolic expansion of the Bezout determinant, and on the evaluation of the roots of the resulting polynomials, and that the solution of both these problems is less efficiently solved as the curve degree increases. In addition, for high degree curves, the implicitization based approach is strongly influenced by increased numerical problems, such that no useful results have yet been obtained for curves of degree greater than 5.

In this paper we introduce new elements to improve the classical implicitization based approach, making it suitable for practical use in a real geometric modeling system. In fact, to avoid the need for symbolic computation, we use a numerical algorithm which solves matrix problems over an integral domain and, in order to reduce the computational complexity, this algorithm is modified to exploit the particular structure implicitly enclosed in the Bezout matrix.

More precisely, we formulate the intersection problem using a Bezout matrix with polynomial entries and we evaluate its numeric–symbolic triangular factorization by means of a generalization to the polynomial ring of the fast fraction-free algorithm proposed in [2] to factorize a Bezout matrix with integer entries. This algorithm, by exploiting the relation existing between Bezout matrices and the Euclidean scheme, succeeds in obtaining a computational complexity of $O(n^2)$.

Finally, to contain the failures introduced by the numerical algorithm, we propose the use of a floating-point, variable-precision arithmetic environment.

This paper is organized as follows. Section 2 introduces the notations needed to define the Bezout resultant of a planar Bézier curve and recall its use in the algebraic approach to the intersection problem. The numerical solution of this problem is considered in Section 3, where an efficient numeric–symbolic approach, based on a fast factorization algorithm over the polynomial ring, is proposed. Numerical examples and comparisons with other standard intersection methods are given in Section 4.

2. Bezout resultant for planar Bézier curve intersection

In the field of *Computer Aided Geometric Design* we are generally concerned with vector-valued functions of one or two variables, i.e. curves and surfaces. In this context, the *Bézier* curve and surface formulations, based on parametric polynomial

representations in the Bernstein basis, provide a powerful framework for efficiently handling the most common geometric problems (see, e.g., [3]).

In the following we will briefly recall the main definitions and properties of the Bernstein/Bézier form that we use to formulate the algebraic approach to the problem of the intersection of two planar rational Bézier curves.

2.1. Basics on Bernstein/Bézier form

Let $\mathbb{P}_n$ be the set of polynomials of degree n. We call $p(t) \in \mathbb{P}_n$ *Bernstein polynomial* if it is given in Bernstein form, namely it is defined by

$$p(t) = \sum_{i=0}^{n} c_i B_i^n(t),$$

where $B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i,\ i = 0, 1, \ldots, n$ are the Bernstein basis functions of degree n on the interval $[0, 1]$.

A *scaled Bernstein polynomial* is defined as a Bernstein polynomial in which the binomial coefficients are absorbed into the polynomial coefficients, that is,

$$p(t) = \sum_{i=0}^{n} \tilde{c}_i (1-t)^{n-i}t^i, \quad \text{where } \tilde{c}_i = \binom{n}{i} c_i.$$

The coefficients $\tilde{c}_i$ are called *scaled Bernstein coefficients*.

Using the parameter substitution $u = t/(1-t)$, we can define the *scaled power form* of the Bernstein polynomial $p(t)$ as

$$\tilde{p}(u) = \sum_{i=0}^{n} \tilde{c}_i u^i. \tag{1}$$

It follows immediately that

$$\tilde{p}(u) = \frac{p(t)}{(1-t)^n}.$$

Two Bernstein polynomials of the same degree n

$$x(t) = \sum_{i=0}^{n} x_i B_i^n(t), \quad y(t) = \sum_{i=0}^{n} y_i B_i^n(t)$$

define a *planar Bézier curve* $P(t)$ with *control points* $C_i = [x_i, y_i]^{\mathrm{T}},\ i = 0, \ldots, n$, namely,

$$P(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \sum_{i=0}^{n} C_i B_i^n(t), \tag{2}$$

which can be seen as an n-degree *vector-valued polynomial* in Bernstein form.

Its *scaled power form* is given by

$$\tilde{P}(u) = \begin{bmatrix} \tilde{x}(u) \\ \tilde{y}(u) \end{bmatrix} = \sum_{i=0}^{n} \tilde{C}_i u^i,$$

where the vector coefficients $\tilde{C}_i = [\tilde{x}_i, \tilde{y}_i]^{\mathrm{T}},\ i = 0, \ldots, n$, are called *scaled control points* of the Bézier curve and $\tilde{x}_i, \tilde{y}_i,\ i = 0, \ldots, n$, are the coefficients of the scaled power form components $\tilde{x}(u), \tilde{y}(u)$.

The use of the scaled power form (1) enables many properties of polynomials in power form to be reproduced for Bernstein polynomials.

For example, the arithmetic operations between two polynomials in Bernstein form can be deduced from those of the power formulation by making use of the scaled Bernstein coefficients [4]. Moreover, it is possible to define the *resultant of a planar Bézier curve*, namely, the resultant of a vector-valued polynomial in Bernstein form, as the resultant of its scaled power form components [5].

More precisely, the Bezout resultant of the Bézier curve $P(t)$ is the determinant of the $n \times n$ symmetric Bezout matrix $\mathscr{B}(P)$, whose entries $r_{i,j}$ are defined by

$$\frac{\tilde{P}(u) \times \tilde{P}(\alpha)}{u - \alpha} := \frac{\begin{vmatrix} \tilde{x}(u) & \tilde{x}(\alpha) \\ \tilde{y}(u) & \tilde{y}(\alpha) \end{vmatrix}}{u - \alpha} = \sum_{i,j=1}^{n} r_{i,j} u^{i-1} \alpha^{j-1}.$$

From the properties of the Bezout matrix, it follows that the entries $r_{i,j}$ can be recursively evaluated in terms of the scaled control points $\tilde{C}_i$ as

$$r_{i,j} = r_{i-1,j+1} + \tilde{C}_{n-i+1} \times \tilde{C}_{n-j}, \quad i = 1, \ldots, n, \quad j = i, \ldots, n, \tag{3}$$

where initially $r_{i,n+1} = 0,\ i = 0, \ldots, n-1,\ r_{0,j} = 0,\ j = 2, \ldots, n+1$ and we assume $\tilde{C}_n \neq 0$.

The value of the Bezout resultant of the Bézier curve $P(t)$ tells us whether the curve has zeros, or equivalently, whether the two polynomial components $x(t)$ and $y(t)$ have common zeros. Furthermore it is possible to evaluate these zeros using the following result (see [5]).

Theorem 1. *Let $P(t)$ be as in* (2) *and $\mathscr{B}(P)$ be its Bezout matrix. If $P(t)$ has exactly k zeros $t_0, t_1, \ldots, t_{k-1}$, then these can be found by performing Gaussian elimination on the rows of $\mathscr{B}(P)$. After elimination, the last non-zero row of $\mathscr{B}(P)$ will be of the form*

$$(0, \ldots, 0, \tilde{h}_k, \ldots, \tilde{h}_0), \quad \tilde{h}_k \neq 0$$

and the zeros of $P(t)$ can simply be obtained by computing the roots of the Bernstein polynomial $h(t) = \sum_{i=0}^{k} h_i B_{i,k}(t)$, with the unscaled coefficients $h_i = \tilde{h}_i / \binom{k}{i}$.

Remark. If $P(t)$ has a zero with multiplicity ℓ at $t = 1$, then it will trivially be $C_{n-\ell-1} = \cdots = C_n = 0$; in this case $\mathscr{B}(P)$ is the matrix $(n-\ell) \times (n-\ell)$ evaluated as in (3) starting from $\tilde{C}_i,\ i = 0, \ldots, n-\ell$ and $\tilde{C}_{n-\ell} \neq 0$.

2.2. Implicitization, inversion and intersection of planar rational Bézier curves

The Bezout matrix of a Bézier curve plays an important role in the solution of some common problems of Computer Aided Geometric Design. Let $P(t),\ t \in [0, 1]$ be a *planar rational Bézier curve* given by

$$P(t) = \begin{bmatrix} x(t)/w(t) \\ y(t)/w(t) \end{bmatrix} = \frac{\sum_{i=0}^{n} C_i w_i B_i^n(t)}{\sum_{j=0}^{n} w_j B_j^n(t)}, \tag{4}$$

where, for $i = 0, \ldots, n,\ C_i = [x_i, y_i]^{\mathrm{T}}$ are the control points of the rational curve and the w_i are positive reals, called weights, such that the n-degree polynomial $w(t) = \sum_{j=0}^{n} w_j B_j^n(t)$ never vanishes for $t \in [0, 1]$ (see [3]). We consider the following two problems:

- *Implicitization*: given a curve defined parametrically in terms of rational polynomials, as in (4), find an implicit polynomial equation $F(x, y) = 0$, which defines the same curve.
- *Inversion*: given the Cartesian coordinates of a point on a parametrically defined curve, find the value(s) of the parameter corresponding to this point.

Both these problems are readily solved using the Bezout matrix. In fact, to solve the implicitization problem for the rational curve (4), we note that for any point $P = (x, y)$ on the curve, we have

$$x = \frac{\sum_{i=0}^{n} x_i w_i B_i^n(t)}{\sum_{j=0}^{n} w_j B_j^n(t)} \quad \text{and} \quad y = \frac{\sum_{i=0}^{n} y_i w_i B_i^n(t)}{\sum_{j=0}^{n} w_j B_j^n(t)}. \tag{5}$$

We therefore define two auxiliary polynomials

$$x_x(t) = \sum_{i=0}^{n} x_i w_i B_i^n(t) - x \sum_{i=0}^{n} w_i B_i^n(t) = \sum_{i=0}^{n} w_i (x_i - x) B_i^n(t),$$
$$y_y(t) = \sum_{i=0}^{n} y_i w_i B_i^n(t) - y \sum_{i=0}^{n} w_i B_i^n(t) = \sum_{i=0}^{n} w_i (y_i - y) B_i^n(t)$$

and consider them as the components of the following Bézier curve

$$P_{xy}(t) = \sum_{i=0}^{n} D_i B_i^n(t),$$

whose control points and their scaled versions are, respectively,

$$D_i = w_i \left(C_i - [x, y]^{\mathrm{T}} \right) \quad \text{and} \quad \tilde{D}_i = \binom{n}{i} D_i, \qquad i = 0, \ldots, n. \tag{6}$$

Then we evaluate, by means of (3), the Bezout matrix of $P_{xy}(t)$. Since the point $P = (x, y)$, given in (5), lies on the curve if and only if the scalar polynomials $x_x(t)$ and $y_y(t)$ have a common root, we have that the implicit equation of the rational Bézier curve is simply the Bezout resultant of the Bézier curve $P_{xy}(t)$, namely,

$$\mathrm{Det}[\mathscr{B}(P_{xy})] = 0.$$

Using the Bezout matrix, we also immediately solve the inversion problem. In fact, given the coordinates (x_0, y_0) of a point along the curve, we can easily evaluate the corresponding parameter value(s) t_0 as the common zero(s) of the polynomials $x_{x_0}(t)$, $y_{y_0}(t)$ by applying Theorem 1 to the Bezout matrix $\mathscr{B}(P_{x_0y_0})$.

Implicitization and inversion also represent two basic steps of another more general problem that can easily be solved by making use of the Bezout resultant. We refer to the following intersection problem:

- *Intersection of planar rational Bézier curves:* given two planar rational Bézier curves $P_1(t)$ and $P_2(s)$ of degree n and m, respectively, given by

$$P_1(t) = \begin{bmatrix} x_1(t)/w_1(t) \\ y_1(t)/w_1(t) \end{bmatrix} = \frac{\sum_{i=0}^{n} C_{1i} w_{1i} B_i^n(t)}{\sum_{j=0}^{n} w_{1j} B_j^n(t)}, \quad t \in [0, 1] \tag{7}$$

and

$$P_2(s) = \begin{bmatrix} x_2(s)/w_2(s) \\ y_2(s)/w_2(s) \end{bmatrix} = \frac{\sum_{i=0}^{m} C_{2i} w_{2i} B_i^m(s)}{\sum_{j=0}^{m} w_{2j} B_j^m(s)}, \quad s \in [0, 1], \tag{8}$$

find both the t and s parameter values and the Cartesian coordinates of all the intersection points within the specified parameter ranges.

The usual algebraic approach to this problem symbolically evaluates the implicit equation of the first curve $P_1(t)$ (implicitization), and then substitutes into this equation the parametric expression of the second curve $P_2(s)$ using

$$x = \frac{x_2(s)}{w_2(s)}, \quad y = \frac{y_2(s)}{w_2(s)}. \tag{9}$$

It turns out that the roots, for $s \in [0, 1]$ of the resulting $(n \cdot m)$-degree polynomial are the parameter values, along the curve $P_2(s)$ of all the intersection points, whose Cartesian coordinates are then obtained from (9). In order to evaluate the corresponding parameter values along the curve $P_1(t)$ it is necessary to solve, for each pair of Cartesian coordinates x_k, y_k, an inversion problem for $P_1(t)$ and check if the corresponding value of t belongs to [0, 1].

3. Numerical solution for the curve intersection problem

The algebraic approach to the intersection problem, that has been outlined in the previous section, consists basically of the following steps:

(1) implicitization of the curve $P_1(t)$ by symbolically evaluating $\mathrm{Det}[\mathscr{B}(P_{1xy})]$;
(2) substitution of the parametric expression of the curve $P_2(s)$ into the implicit equation of the curve P_1;
(3) evaluation of the roots of the resulting polynomial equation, for $s \in [0, 1]$;
(4) solution of as many inversion problems for the curve $P_1(t)$ as these are roots found in the previous step.

This approach is difficult to realize in a real CAGD system, mainly due to its need for symbolic computation. To avoid this necessity, that is, to provide a numerical solution to the algebraic approach to the curve intersection problem, it is necessary to interchange step 2 with step 1, and to construct the Bezout matrix of the Bézier curve whose components are polynomials in the variable t with polynomial coefficients in the variable s, namely of $P_{1x_2(s),y_2(s),w_2(s)}$.

More precisely, starting from the curve $P_1(t)$, according to (6) we set

$$\tilde{D}_\ell = \binom{n}{\ell} w_{1\ell} \begin{bmatrix} x_{1\ell} - x \\ y_{1\ell} - y \end{bmatrix}, \quad \ell = 0, \ldots, n$$

and, using (9), we substitute the coordinates (x, y) with the parametric components of the curve $P_2(s)$, obtaining

$$\tilde{D}_\ell(s) = \frac{1}{w_2(s)} \binom{n}{\ell} w_{1\ell} \begin{bmatrix} x_{1\ell} w_2(s) - x_2(s) \\ y_{1\ell} w_2(s) - y_2(s) \end{bmatrix}. \tag{10}$$

In this way the building blocks $\tilde{D}_\ell(s) \times \tilde{D}_k(s)$, which are used, according to (3), to compute the $n \times n$ Bezout matrix $\mathscr{B}(P_{1x_2(s),y_2(s),w_2(s)})$ are given, except for a non-null scaling factor $1/w_2(s)$, by

$$\begin{aligned} \tilde{D}_\ell(s) \times \tilde{D}_k(s) = w_{1\ell} w_{1k} \binom{n}{\ell} \binom{n}{k} \sum_{i=0}^{m} [&(y_{1\ell} - y_{1k}) x_{2i} + (x_{1k} - x_{1\ell}) y_{2i} \\ &+ (x_{1\ell} y_{1k} - x_{1k} y_{1\ell})] w_{2i} B_i^m(s), \end{aligned} \tag{11}$$

and the entries $r_{ij}(s)$, $i, j = 1, \ldots, n$ of the Bezout matrix $\mathscr{B}(P_{1x_2(s),y_2(s),w_2(s)})$ turn out to be m-degree polynomials in Bernstein form.

In order to obtain the s parameter values of the intersection points, it is necessary to evaluate the determinant of the polynomial matrix $\mathscr{B}(P_{1x_2(s),y_2(s),w_2(s)})$, which will be a $(m \cdot n)$-degree polynomial, and to compute all its real roots for $s \in [0, 1]$. Step 4 will then give the corresponding t parameter values.

This new form of the curve intersection algorithm is more suited to a numerical approach but requires the use of efficient procedures for the accurate solution of the following numerical problems:

(a) the evaluation of the determinant of a matrix with polynomial entries;
(b) the evaluation of the K real roots, in $[0, 1]$, of the resulting $(m \cdot n)$-degree polynomial;

(c) the realization of Gaussian elimination on the rows of the Bezout matrix for each valid root s_k, for $k = 1, \ldots, K$.

This paper is mainly concerned with a fast numerical algorithm, that solves the first and the third problem at the same time and, because of its feature of working over an integral domain, maintains all the advantages of symbolic computation.

3.1. Fast triangularization over an integral domain

It is well known that the triangular factorization of a Bezout matrix of order n can be evaluated by means of fast algorithms, i.e. requiring $O(n^2)$ arithmetic operations. This is due to the particular structure that the Bezout matrix presents implicitly enclosed in its entries (see [2,6,9] for details).

In particular, we refer to the algorithm proposed by Bini–Gemignani for the triangularization of a Bezout matrix with entries over the integral domain $\mathbb{Z}$. It consists of a fast procedure which reduces the matrix into triangular form by means of ring operations and exact divisions only. This algorithm represents an improvement of the Bareiss fraction-free elimination scheme [1], based on the property that the structural invariance of the Bezout matrix under Schur complementation still holds if the Bareiss variant of Gaussian elimination is applied. This structure preserving property has allowed the authors to reduce the computational complexity of the Bareiss algorithm from $O(n^3)$ to $O(n^2)$ arithmetic operations, without losing the characteristic of keeping at a minimum level the growth of the length of the integers involved in the computations, namely $O(n \log nc)$ bits, where c is an upper bound of the moduli of the integer matrix entries.

In order to realize an efficient procedure to solve problems (a) and (c) of the previous subsection, we have extended the Bini–Gemignani algorithm to the more general case of a Bezout matrix with polynomial entries, achieving in this way a kind of numeric–symbolic factorization.

More precisely, let $\mathscr{B}$ be the $n \times n$ Bezout matrix, whose entries $r_{i,j}(s)$ are m-degree scaled Bernstein polynomials, as mentioned in the previous section. We have realized the following version of the Bini–Gemignani factorization algorithm, which, by identifying Berstein polynomials with their coefficient vectors, yields the polynomial entries of the upper triangular factor and gives an explicit expression for the (polynomial) determinant of the matrix. For the sake of simplicity, we describe the algorithm in the case of a strongly non-singular matrix $\mathscr{B}$.

3.2. Fast factorization algorithm

- *input*: control points and weights of $P_1(t)$ and $P_2(s)$ as given by (7) and (8);
- *initialization*: the scaled Bernstein polynomial entries of the first two rows of $\mathscr{B}$ are obtained by applying the recursive scheme given in (3) and using (11):

$$r_{1,j}(s) = \tilde{D}_n(s) \times \tilde{D}_{n-j}(s), \quad j = 1, \ldots, n$$
$$r_{1,n+1}(s) = 0$$
$$r_{2,j}(s) = r_{1,j+1}(s) + \tilde{D}_{n-1}(s) \times \tilde{D}_{n-j}(s), \quad j = 2, \ldots, n$$

- *computation:*

$$r_{0,0}(s) = 1$$
$$for\ i = 1, \ldots, n-1$$
$$(1)\begin{cases} for\ j = i+1, \ldots, n \\ \quad r_{i+1,j}(s) = \frac{r_{i,i}(s)r_{i+1,j}(s) - r_{i,i+1}(s)r_{i,j}(s)}{r_{i-1,i-1}(s)} \end{cases}$$
$$if\ i+1 < n$$
$$(2)\begin{cases} r_{i+1,n+1}(s) = 0 \\ for\ j = i+2, \ldots, n \\ \quad r_{i+2,j}(s) = r_{i+1,j+1}(s) + \frac{r_{i+1,j}(s)r_{i,i+1}(s) - r_{i+1,i+1}(s)r_{i,j}(s)}{r_{i,i}(s)} \end{cases}$$

- *output*: the polynomial entries of the upper triangular factor. In particular $r_{n,n}(s)$ is an $(m \cdot n)$-degree scaled Bernstein polynomial, corresponding to the determinant of $\mathscr{B}$.

In the above scheme, loop (1) represents the Bareiss elimination step, while loop (2) evaluates the second row of the new Schur complement. Of course, in the general case, the algorithm uses a pivoting strategy which is applied whenever an $r_{i,i}(s) = 0$ is encountered, and the factorization steps are modified accordingly (see [2]).

Concerning the cost analysis for evaluating and factorizing the Bezout matrix $\mathscr{B}$, it is immediately clear that, in the strongly non-singular case, it consists of $O(n^2)$ arithmetic operations between polynomials in Bernstein form. When the matrix is not strongly non-singular, and therefore a pivoting strategy is needed, this cost slightly increases but still remains of order $O(n^2)$, as shown in [2].

In order to realize these operations with improved numerical stability, with respect to the power form, but at the same computational cost, we have used their scaled coefficients, according to the following rules:
if $a(s)$ and $b(s)$ are two scaled Bernstein polynomials of exact degree k and h, respectively, $k \geqslant h$, then, in the algorithm:

- sums and differences are performed only between polynomials of the same degree (i.e. $k = h$), and their coefficients are simply computed by:

$$\tilde{c}_i = \tilde{a}_i \pm \tilde{b}_i, \quad i = 0, \ldots, k;$$

- multiplications are realized according to the usual convolution rule, that is, the coefficients of the product are given by:

$$\tilde{c}_i = \sum_{j=\max(0,i-h)}^{\min(k,i)} \tilde{a}_j \cdot \tilde{b}_{i-j}, \quad i = 0, \ldots, k+h$$

with a computational cost of $(k+1)(h+1)$ arithmetic multiplications;

- divisions are only exact and the results are polynomials of degree $k-h$. Their coefficients are therefore evaluated by solving an upper triangular linear system; using backward substitution, it yields:

$$\tilde{c}_{k-h} = \frac{\tilde{a}_k}{\tilde{b}_h}$$
$$\tilde{c}_{i-h} = \frac{1}{\tilde{b}_h}\left(\tilde{a}_i - \sum_{j=i-h+1}^{\min(k-h,i)} \tilde{b}_{i-j}\tilde{c}_j\right), \quad i = k-1, \ldots, h;$$

the corresponding computational cost is $(k - \frac{3}{2}h + 1)(h+1)$ arithmetic multiplications/divisions.

It follows that the cost of the whole algorithm depends on the growth of the degrees of the polynomials involved in the computation. In the integer case, this growth has been shown to be of order $O(n \log nc)$. This order is mantained in our realization, with the additional feature that we are able to give an exact evaluation of the constant c. In fact, if the degree of the polynomial entries of $\mathscr{B}$ is m, the maximum degree reached during the computation is exactly $2(n-1)m$. Hence, the overall arithmetic cost is $O(n^2 \nu(2(n-1)m))$, where $\nu(k)$ denotes the arithmetic cost of multiplying k-degree polynomials.

Another advantage of our numeric–symbolic approach is that the solutions of the necessary inversion problems of step (c) are simply obtained. In fact, for each root s_k of the polynomial $r_{n,n}(s)$, it is only necessary to evaluate in s_k the two polynomial entries of the $(n-1)$th row of the upper triangular factor. According to Theorem 1, if both polynomials do not vanish in s_k, the unique corresponding value t_k is immediately obtained. Otherwise, it is necessary to repeat the procedure on the $(n-2)$th row, and so on.

More precisely, we have:

- *Inversion:*
  1. for each root s_k, $k = 1, \ldots, K$ of $r_{n,n}(s) = 0$ do
     - 1.1. set $u_k := s_k/(1-s_k)$;
     - 1.2. set $\ell := n-1$;
     - 1.3. for $j = \ell, \ldots, n$ evaluate in u_k the scaled power form (1) of the $m\ell$-degree polynomias $r_{\ell,j}(s)$, and set $\tilde{h}_{n-j} := \tilde{r}_{\ell,j}(u_k)$;
     - 1.4. for $j = \ell, \ldots, n$, if $\tilde{h}_{n-j}$ do not vanish, compute the t_k value(s), according to Theorem 1, otherwise update $\ell := \ell - 1$ and continue from 1.3;
     - 1.5. if $t_k \notin [0, 1]$ discard this point;

2. compute the (x_k, y_k) Cartesian coordinates of the intersection points by evaluating $P_1(t_k)$ or $P_2(s_k)$ for $k = 1, \ldots, K$.

In the most frequent case, namely when the K intersection points are simple, it follows that the computational cost of the first step of the inversion problem consists of $2K(n-1)m$ arithmetic multiplications.

4. Examples and numerical results

The proposed algorithm has been extensively tested by considering the intersection of many rational Bézier curves. In order to realize the whole intersection algorithm we have added to our factorization and inversion procedures a root evaluation algorithm, taken from the literature, that guarantees an optimal behaviour with respect to the numerical stability for high degree polynomials. More precisely, we have implemented the Bézier Clipping root finder method proposed in [10]. This method is based on the characteristics of the Bernstein/Bézier form and its use has been possible as our intersection algorithm only deals with polynomials in Bernstein form.

According to the numerical considerations deriving from this experimentation, some representative examples have been chosen, consisting of curves of different degrees and numbers of intersections, showing the performance of the algorithm. Timing comparisons, given in Table 1, have been run against our implementation of the *Bézier Clipping curve intersection algorithm*. This method, proposed in [12], exploits the Bézier Clipping technique, already used in the literature for root finder and ray-patch algorithms, to realize an intelligent interval Newton method, in which geometric insight is used to identify regions of the parameter domain which exclude the solution set. The choice of the Bézier Clipping curve intersection algorithm is motivated by the analysis given in [12] that shows that this method seems to be faster than conventional curve intersection methods.

Both methods have been implemented in ANSI-C language on a PC-Linux. Timing tests were run using double precision arithmetic, and computing the answers to eight decimal digits of accuracy.

Table 1
Time comparisons (milliseconds); for the proposed algorithm, the factorization, root evaluation and inversion times are explicitly given in the last three columns

Example no.	Degrees (n, m)	No. of integers	Bézier Clipping Total	Our proposal			
				Total	Factorization	Root evaluation	Inversion
1	3,3	9	0.388	0.214	0.018	0.172	0.024
2	3,4	3	0.151	0.117	0.026	0.081	0.010
3	3,6	6	0.419	0.406	0.045	0.333	0.028
4	3,5	3	0.184	0.173	0.035	0.127	0.011
5	5,5	3	0.312	0.709	0.209	0.483	0.017

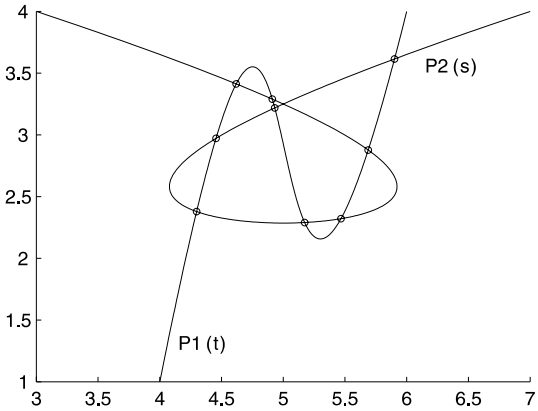

Fig. 1. Curves of example no. 1, with nine intersection points.

The first example is taken from [11]; it is a test example for several intersection procedures given in the literature. It consists of two rational Bézier curves of degree 3 which, as shown in Fig. 1, present nine intersection points. Their control points and weights are:

C_1	4.0	5.0	5.0	6.0	C_2	7.0	1.0	9.0	3.0
	1.0	6.0	0.0	4.0		4.0	2.0	2.0	4.0
w_1	1.0	2.0	2.0	1.0	w_2	1.0	2.0	2.0	1.0

In this case, the proposed algorithm computes the nine intersection points with the desired accuracy and a time reduction, compared to the Bézier Clipping, of almost 40%.

The second and third examples have been chosen to show the behaviour of the proposed algorithm for the intersection of two Bézier curves whose control points are real numbers that are not exactly representable using the considered floating-point precision. Example no. 2 consists of the non-rational curves of degrees 3 and 4, respectively, shown in Fig. 2, which present three intersection points. Their control points and weights are:

C_1	−0.2	−0.1	0.0	0.1	C_2	−0.2	−0.4	0.0	0.4	0.2
	0.0	0.9	−0.9	0.0		0.5	0.1	0.1	−0.1	−0.5
w_1	1.0	1.0	1.0	1.0	w_2	1.0	1.0	1.0	1.0	1.0

Example no. 3 consists of two non-rational curves of degree 3 and 6 respectively. The curve of degree 3 is the same as example no. 2, while the control points and weights of the 6-degree curve are:

C_2	−0.5	0.6	−0.7	0.8	0.1	0.1	−0.5
	0.5	−0.1	−0.1	−0.8	0.7	−0.6	0.5
w_2	1.0	1.0	1.0	1.0	1.0	1.0	1.0

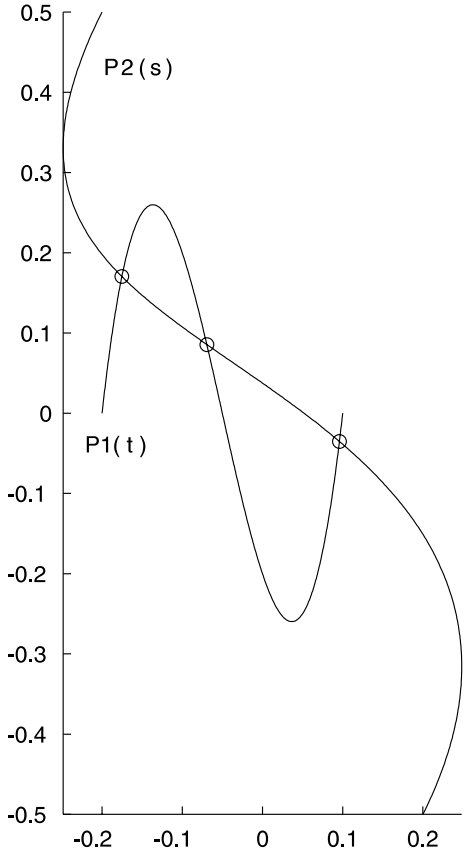

Fig. 2. Curves of examples no. 2, with three intersection points.

The two curves are shown in Fig. 3 and present six intersection points. In both cases, the right intersection points are evaluated, but in some intersections we lose some digits of accuracy, especially in the third example. Regarding the timing comparison, the proposed algorithm in the third example is only slightly faster than the Bézier Clipping, but, as shown in Table 1, this mainly depends on the time needed to compute the six roots of the 18-degree resultant polynomial.

We have therefore considered examples where the control points are integer numbers or real numbers which can be exactly represented in the floating-point precision taken into consideration. This choice has allowed us to completely exploit the characteristic that the factorization algorithm works over integral domains. In fact, this guarantees that the factorization algorithm is performed in exact floating-point arithmetic, and that the growth of the numbers involved in the computation is controlled. In fact, as we have experimentally verified, also for the polynomial coefficients a length bound of the kind $O(n \log nc)$ bits still holds.

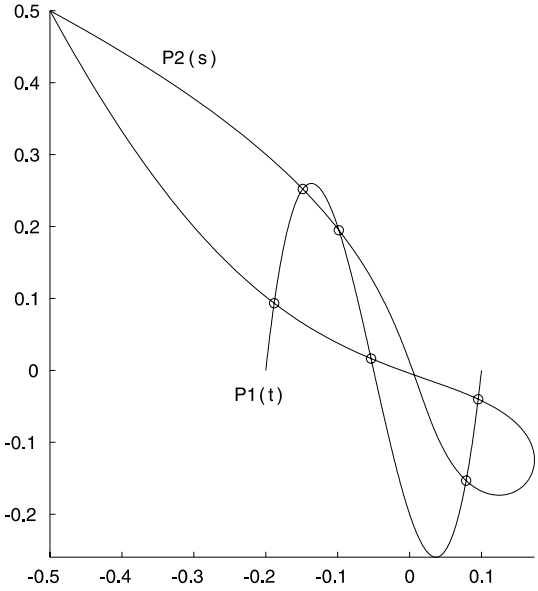

Fig. 3. Curves of example no. 3, with six intersection points.

Example no. 4 consists of two rational Bézier curves, given in Fig. 4, whose control points and weights are real numbers exactly rapresentable in the given floating-point precision.

C_1	-0.25	-0.25	0.0	0.25
	0.0	1.0	-1.0	0.0
w_1	1.0	1.0	1.0	1.0

C_2	-0.25	-0.5	-0.25	-0.25	0.5	-0.25
	0.5	0.25	0.25	-0.25	-0.25	-0.5
w_2	1.0	1.0	1.0	1.0	2.0	1.0

The algorithm evaluates the intersection points with the required accuracy and a computational time which is slightly less than that of the Bézier Clipping method.

In example no. 5 we consider two rational curves of degree 5 with integer control points and suitable real weights. These are:

C_1	-1.0	0.0	-1.0	0.0	1.0	1.0
	1.0	-1.0	-1.0	1.0	1.0	-1.0
w_1	1.0	0.2	0.2	0.2	0.4	1.0

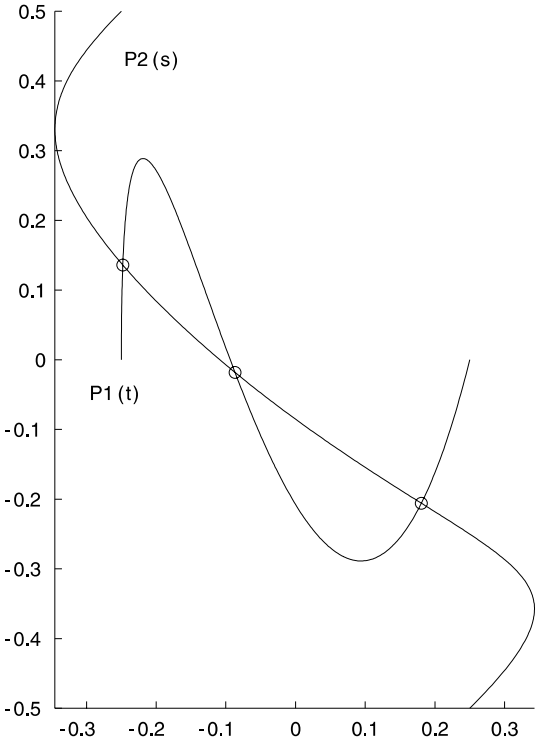

Fig. 4. Curves of example no. 4, with three intersection points.

C_2	-1.0	-1.0	0.0	0.0	1.0	1.0
	0.0	1.0	1.0	-1.0	-1.0	0.0
w_2	1.0	0.4	0.2	0.2	0.2	1.0

This choice leads to a Bezout matrix with integer coefficient polynomial entries. The two curves are shown in Fig. 5 and present three intersection points. Also in this case, the intersection points are evaluated with the required precision, but the computational time is significantly higher than that of the Bézier Clipping method. Nevertheless, as is clearly shown in Table 1, most of the time of this run has been spent on obtaining the solution of the polynomial equation $p(s) = 0$.

This example gives us the opportunity to make a general consideration: by increasing the curve degrees, the part of the intersection algorithm that is most time-consuming is the solution of the polynomial equation $p(s) = 0$. Therefore, a suitable choice of a quick root finder could result in a significant decrease of the total execution time. We have not optimized the proposed intersection algorithm from this point of view. Instead, our main goal has been the optimization of the factorization and inversion phases, both as regards computing time and as regards accuracy.

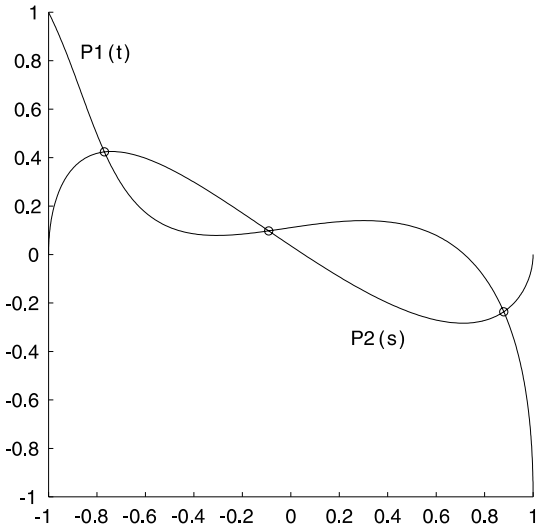

Fig. 5. Curves of example no. 5, with three intersection points.

Nevertheless, even using our numeric–symbolic approach, there is a natural limit to the accuracy of the coefficients of the final polynomial; in fact, computing the roots of high degree polynomials may be an ill-conditioned problem and this yields an upper limit to the degrees of the curves to be intersected. In our experimentation this limit is reached in example no. 5, but it can easily be extended using suitable multiple-precision floating-point arithmetic.

This is shown in the following example where we consider the two rational curves of degrees 5 and 8 shown in Fig. 6, whose control points and weights are exactly representable in double floating-point precision, namely:

C_1	-0.5	0.0	-0.5	0.0	0.5	0.5
	0.5	-0.5	-0.5	0.5	0.5	-0.5
w_1	1.0	0.5	0.25	0.25	0.5	1.0

C_2	0.0	-3.0	6.5	-7.0	0.0	7.0	-6.5	3.0	0.0
	-0.25	4.0	-17.0	35.0	-43.0	35.0	-17.0	4.0	-0.25
w_2	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

From the implicitization of the first curve, we obtain a Bezout matrix with 8-degree polynomial entries whose coefficients are exactly representable with 20 bits. In order to obtain a final polynomial with exact coefficients, and bearing in mind, since in this case $n = 5$ and $c = 2^{20}$, the growth of $O(n \log nc)$ during the computation, the required precision turns out to be greater than 100 bits. We have therefore used the

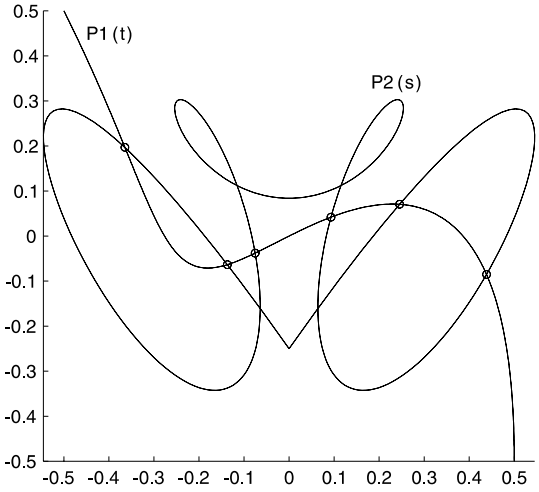

Fig. 6. Curves of example no. 6, with six intersection points.

GNU Multiple Precision arithmetic library to provide the desired precision and our algorithm has computed all the intersection points within the accuracy imposed as the stop test.

5. Conclusions

We have presented an efficient algorithm for the numerical solution of the algebraic approach to rational Bézier curve intersection. It is based on the use of a fast factorization algorithm over the polynomial ring, and on the characteristic of operating with polynomials in Bernstein form. By identifying Bernstein polynomials with their coefficient vectors and performing ring operations and exact divisions only, the algorithm is capable of accurately evaluating the intersection of planar rational Bézier curves of any degree, avoiding the need for symbolic computation. However, it requires the use of a suitable multiprecision floating-point environment and a fast and robust procedure for the evaluation of real roots, within a given interval, of high degree polynomials. Concerning the first point, the GNU Multiple Precision arithmetic library could represent the best choice, as it provides optimized tools for performing floating-point arithmetic with arbitrarily high precision, while the second point still needs more future work.

Acknowledgements

We are grateful to the reviewers for their comments.

References

[1] E. Bareiss, Sylvester's identity and multistep integer preserving Gaussian elimination, Math. Comp. 22 (1968) 565–578.
[2] D.A. Bini, L. Gemignani, Fast fraction-free triangularization of Bezotians with applications to sub-resultant chain computation, Linear Algebra Appl. 284 (1998) 19–39.
[3] G. Farin, Curves and Surfaces for CAGD, A practical guide, third ed., Academic Press, 1993.
[4] R.T. Farouki, V.T. Rajan, Algorithms for polynomials in Bernstein form, Comput. Aided Geom. Design 5 (1988) 1–26.
[5] R.N. Goldman, T.W. Sederberg, D.C. Anderson, Vector elimination: a technique for the implicitization, inversion, and intersection of planar parametric rational polynomial curves, Comput. Aided Geom. Design 1 (1984) 327–356.
[6] G. Heinig, K. Rost, Algebraic Methods for Toeplitz-like Matrices and Operators, Birkaeuser, 1984.
[7] C.M. Hoffmann, Geometric and Solid Modeling. An Introduction, Morgan Kaufmann Publishers, 1989.
[8] J. Hoschek, D. Lasser, Fundamentals of Computer Aided Geometric Design, A.K.Peters, 1993.
[9] H. Levi-Ari, Y. Bistriz, T. Kailath, Generalized Bezoutians and familes of efficient zero location procedures, IEEE Trans. Circuits and Systems 38 (1991) 170–186.
[10] T. Nishita, T.W. Sederberg, M. Kakimoto, Ray Tracing trimmed rational surface patches, Comput. Graphics 24 (4) (1990) 337–345.
[11] T.W. Sederberg, D.C. Anderson, R.N. Goldman, Implicitization, Inversion, and intersection of planar rational curves, Comput. Vision Graphics Image Process. 31 (1985) 89–102.
[12] T.W. Sederberg, T. Nishita, Curve intersection using Bézier Clipping, Comput. Aided Design 22 (9) (1990) 538–546.